Trabajo Práctico 1
Sistema de archivos FAT16 en lenguaje ensamblador

Organización del Computador 2

1er. Cuatrimestre 2010
Versión 3

1. Introducción teórica

En las viejas epocas de Windows 95, aparecieron los discos rígidos de 2 Gb o más. Para poder soportarlos, los sistemas operativos debieron permitir que sus sistemas de archivos soporten estas enormes cantidades de datos. Para ello, Micro$oft desarrolló su sistema de archivos FAT32. Como en esta materia somos tan modernos, vamos a realizar algunas operaciones con el sistema de archivos FAT16, o sea, su predecesor.

1.1. Estructura

FAT16 o *File Allocation Table 16* es un sistema de archivos desarrollado a partir de FAT12 en el año 1987. Este sistema de archivos ve el medio de almacenamiento como un arreglo de *clusters*. Cada clúster puede estar ocupado por una porción (o el total) de un archivo o directorio. Se utiliza la tabla FAT para encontrar cada uno de estos clústeres.

El medio de almacenamiento se organiza en tres áreas principales:

- El sector de booteo.
- La tabla de asignación de archivos (FAT)
- El área de directorio y datos.

1.1.1. Sector de Booteo

El sector de booteo ocupa un sector, y siempre se ubica en el sector 0. Si el medio de almacenamiento no esta particionado, es el primer sector del mismo.

Este sector contiene código y datos mezclados. La siguiente tabla muestra el contenido de este sector:

Offset	Tamaño	Significado
0	3	Primeros 3 bytes. Usados para saltar al código de booteo.
3	8	Identificador OEM. “mkdosfs”
11	2	Número de Bytes por sector.
13	1	Número de sectores por clúster.
14	2	Número de sectores reservados. Incluye al sector de booteo.
16	1	Número de FATs. Normalmente es 2.
17	2	Número de entradas en el directorio raíz.
19	2	Número total de sectores en el volumen. Si es 0, entonces hay mas de 65535 sectores.
21	1	*Media Descriptor Type*. Describe el tipo de medio de almacenamiento.
22	2	Número de sectores por FAT.
24	2	Número de sectores por pista.
26	2	Número de cabezas o lados en el medio de almacenamiento.
28	4	Número de sectores ocultos.
32	4	Número de sectores en el medio. Este se utiliza sólo si hay más de 65535 sectores en el volumen.
36	1	Número de disco.
37	1	Reservado.
38	1	Firma. Debe ser 0x28 o 0x29).
39	4	Número de serie del volumen (VolumeID).
43	11	Etiqueta del volumen.
54	8	*String* identificador del sistema
62	448	Código de booteo.
510	2	Firma de partición booteable: 0xAA55.

1.1.2. FAT (*File Allocation Table*)

Esta tabla se ubica en el dispositivo de almacenamiento y se utiliza para indicar la ubicación y estado de todos los clústers de datos en el dispositivo. Puede ser considerada como la “tabla de contenidos”. Un clúster puede estar disponible, reservado para el sistema operativo, no disponible por contener un sector malo del disco o puede estar en uso por un archivo. Los clústeres de un archivo no necesariamente deben estar uno a continuación del otro. La idea es que esta tabla permita seguir la listas de clústeres que componen un archivo. FAT 16, como su nombre lo indica, utiliza 16 bits para indexarlos.

Valores posibles de una entrada de la FAT:

Valor	Significado
`0x0000`	Clúster libre
`0x0001-0x0002`	No permitido
`0x0003-0xFFEF`	Número del próximo clúster
`0xFFF8-0xFFFF`	Fin del archivo

Notas sobre la numeración de los clústeres:

- Los clústeres se numeran del 0 en adelante.
- El primer clúster de datos tiene el índice 2.
- Las primer entrada de la FAT indica el tipo de medio. Debe ser el mismo que se indica en el sector de booteo.
- La segunda entrada de la FAT indica el estado. `0xFFFF` indica que está limpia. `0xF7FF` indica que esta montada o en uso.

1.1.3. Área de directorios y datos

Un directorio simplemente guarda la información del nombre, sus atributos y donde está el primer clúster del directorio o archivo. FAT16 usa el formato de nombre estándar 8.3. Ocho caractéres para el nombre y tres para la extensión.

Formato de una entrada de directorio:

Offset	Tamaño	Significado
0	11	Nombre de archivo en formato 8.3.
11	1	Atributos: ver Byte de atributo.
12	1	Reservado.
13	1	Décimas de segundo del tiempo de creación.
14	2	Tiempo de creación. 5 bits para la hora, 6 para los minutos, 5 para los segundos.
16	2	Fecha de creación. 7 bits para el año, 4 para el mes, 5 para el día.
18	2	Última fecha de acceso.
20	2	Cero.
22	2	Última fecha de modificación.
24	2	Último tiempo de modificación.
26	2	Primer clúster del archivo.
28	4	Tamaño del archivo en bytes.

Byte de atributo:

7	6	5	4	3	2	1	0
Res	Res	Archivo	Directorio	VolumeID	Sistema	Oculto	Sólo Lectura.

Nota: Res = Reservado.

El bit *archivo* no indica que es un archivo. Este se utiliza en sistemas de backup. Para determinar si es archivo o directorio, hay que ver el bit *directorio*.

Notas sobre el campo *nombre*:

El primer *byte* es tratado de forma especial. Estas son las reglas que se aplican:

`0x00` se interpreta como "detener la búsqueda, no hay mas entradas en este directorio".

`0x20` no debe existir.

`0xe5` se interpreta como una entrada libre.

Todas los nombres tienen exactamente 11 bytes. Para rellenarlos se usa el caracter `0x20` (espacio).

Con el paso del tiempo, el sistema FAT16 original se modificó para soportar nombres largos. Para ello, las entradas de directorio se empezaron a utilizar para algo mas que representar archivos y directorios. Hay entradas que contienen el nombre en formato *Unicode* y un campo que indica a que entrada del directorio corresponde ese nombre largo. Para no confundirlas con entradas de archivo o directorios, los valores del campo *atributos* y *primer clúster del archivo* tienen los valores `0x0F` y `0x0000` respectivamente. Valores que no tienen sentido dentro de una entrada de directorio. Para el objetivo de este trabajo práctico no es necesario tener en cuenta los nombres largos. Estas entradas deben ser obviadas.

1.2. Funcionamiento

Cada entrada de la tabla FAT está vinculada a un clúster. Este vínculo es biyectivo y los índices se corresponden. Es decir, la primer entrada de la FAT corresponde al valor del primer clúster, la segunda entrada de la FAT corresponde al valor del segundo clúster, etc.

No todos los archivos o directorios van a ocupar necesariamente un clúster. FAT16, cuando necesita ocupar otro clúster, hace uso de la tabla FAT para encontrar alguno libre, y escribe, en el valor que correponde al anterior, cual es el próximo. De esta forma, cuando uno llega al final de un clúster, verifica en la entrada FAT correspondiente si el archivo terminó, o hay un clúster siguiente, y cuál es.

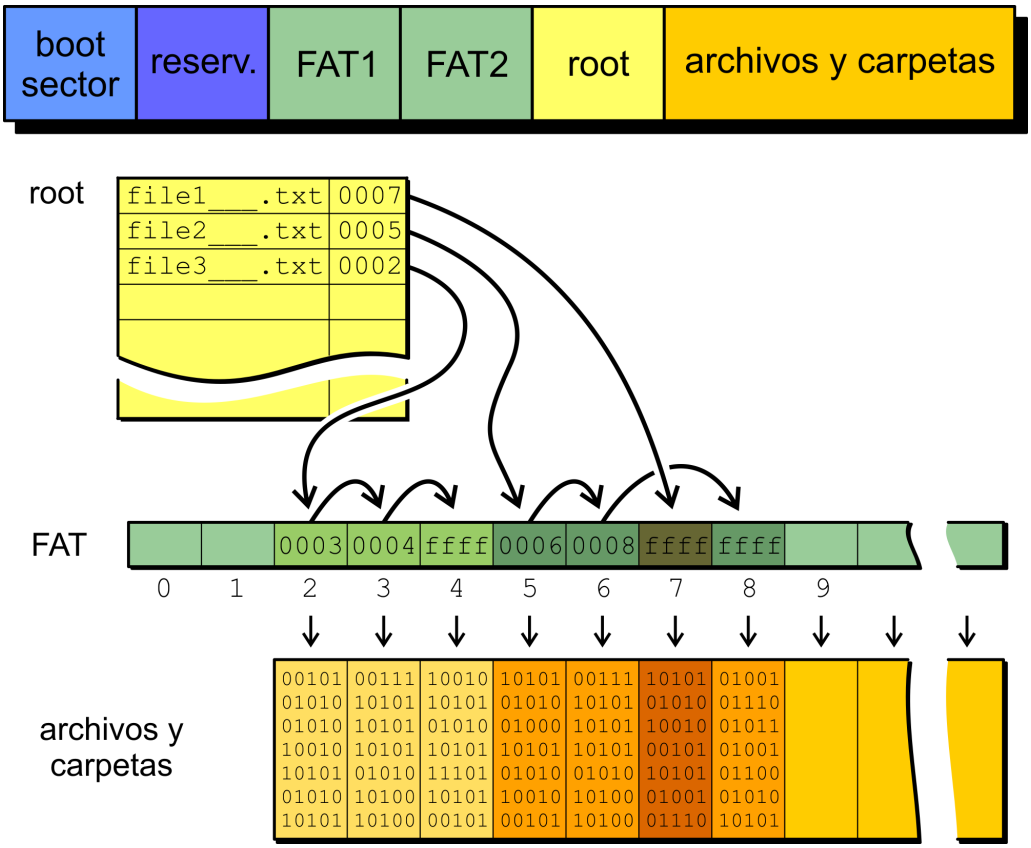

Figura 1: Esquema del funcionamiento de FAT16

1.3. Funciones auxiliares

Para lograr el objetivo, es necesario la utilización de algunas funciones auxiliares provistas por las librerías de C.

1.3.1. `open`

```
int open(const char *filename, int flags);
```

Esta función devuelve un descriptor de archivo (o *file descriptor*) correspondiente al archivo `filename`. Si ocurre un error, devuelve `-1`.

Los flags pueden ser:

- `O_RDONLY`: Sólo lectura.
- `O_WRONLY`: Sólo escritura.
- `O_RDWR`: Lectura/escritura.

- `O_CREAT`: Si no existe, lo crea.

El parámetro `mode` debe ser especificado si se usa el flag `O_CREAT`. Se utiliza para indicar los permisos con los que se va a crear el archivo. Puede ser una combinación de los siguientes:

- `S_IRWXU`: Lectura/Escritura/Ejecución del *owner*.
- `S_IRUSR`: Lectura del usuario.
- `S_IWUSR`: Escritura del usuario.
- `S_IXUSR`: Ejecución del usuario.
- `S_IRWXG`: Lectura/Escritura/Ejecución del grupo.
- `S_IRGRP`: Lectura del grupo.
- `S_IWGRP`: Escritura del grupo.
- `S_IXGRP`: Ejecución del grupo.
- `S_IRWXO`: Lectura/Escritura/Ejecución de otros.
- `S_IROTH`: Lectura de otros.
- `S_IWOTH`: Escritura de otros.
- `S_IXOTH`: Ejecución de otros.

1.3.2. `write`

```
ssize_t write(int fd, const void *buf, size_t count);
```

Esta función escribe `count` bytes en el archivo descripto por `fd` obtenidos a partir de `buf`. Devuelve la cantidad de bytes escritos.

1.3.3. `fstat`

```
int fstat(int filedes, struct stat *buf);
```

Esta función recibe un descriptor de archivo `filedes` y rellena el *struct* `buf` con los datos que se indican a continuación.

```
struct stat {
    dev_t     st_dev;     /* ID of device containing file */
    ino_t     st_ino;     /* inode number */
    mode_t    st_mode;    /* protection */
    nlink_t   st_nlink;   /* number of hard links */
    uid_t     st_uid;     /* user ID of owner */
    gid_t     st_gid;     /* group ID of owner */
    dev_t     st_rdev;    /* device ID (if special file) */
```

```
    off_t     st_size;    /* total size, in bytes */
    blksize_t st_blksize; /* blocksize for filesystem I/O */
    blkcnt_t  st_blocks;  /* number of blocks allocated */
    time_t    st_atime;   /* time of last access */
    time_t    st_mtime;   /* time of last modification */
    time_t    st_ctime;   /* time of last status change */
};
```

Devuelve `0` si tuvo éxito o `-1` si ocurrió algún error.

1.3.4. `close`

```
int close(int fd);
```

Esta función cierra el archivo descripto por `fd`. Devuelve `0` si tuvo éxito o `-1` si ocurrió algún error.

1.3.5. `mmap`

```
void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);
```

Crea un mapeo a memoria a partir de la dirección `addr` del archivo descripto por `fd`. El mapeo es de tamaño `length` y comienza a partir de `offset`. El parámetro `prot` indica si hay algún tipo de protección de memoria y `flags` se utiliza para indicar características del mapeo.

Si `addr` es `NULL` entonces el kernel elige la dirección a partir de la cual se realiza el mapeo y la devuelve en `addr`.

El parámetro `prot` debe tener alguno de los siguientes valores:

- `PROT_EXEC`: Se puede ejecutar el código.
- `PROT_READ`: Se puede leer.
- `PROT_WRITE`: Se puede escribir.
- `PROT_NONE`: No hay protección.

Devuelve la dirección del mapeo si tuvo éxito. `MAP_FAILED` si hubo error.

1.3.6. `munmap`

```
int munmap(void *addr, size_t length);
```

Deshace lo hecho con `mmap`.

Devuelve `0` si tuvo exito o `-1` si ocurrió algún error.

1.3.7. Valores de las constantes y tamaños de los tipos

Dependiendo del sistema operativo elegido para realizar este trabajo, puede ser que se encuentren que los valores de algunas constantes difieran o que los tamaños de los tipos de datos utilizados varíen. Para ello se da una herramienta (`sizes.c`) que, luego de compilarla y ejecutarla, nos muestra por pantalla los tamaños de distintos tipos de datos y constantes necesarias para este trabajo práctico.

Es muy importante que en el informe aclaren qué SO utilizaron, y sobre qué procesador.

1.3.8. Creación de imágenes de prueba

Imagen vacía

Para poder probar la aplicación, es necesario contar con imágenes de prueba. La forma de crearlas, bajo linux, es la siguiente:

```
dd if=/dev/zero of=./imagen.bin bs=1024 count=TAMAÑO
```

Este comando crea un archivo `imagen.bin` lleno de ceros. Hay que reemplazar `TAMAÑO` por la cantidad de kb que queremos que tenga.

Formateo de imagen

El próximo paso es darle formato a la imagen.

```
mkfs.vfat -F16 imagen.bin
```

Este otro comando le da formato FAT16 al archivo `imagen.bin`.

Otros parámetros que pueden ser agregados son:

- `-f N`: Crea `N` copias de la tabla FAT.
- `-n ORGA2`: Establece `ORGA2` como la etiqueta del medio.

Copiado de archivos

Para modificar la imagen se pueden hacer dos cosas: montarla o utilizar `mtools` para modificarla. La opción mas sencilla es montarla, pero esto requiere acceso como superusuario.

Opcioñ A: montado de imagen.

```
sudo mount -o loop -t vfat imagen.bin DESTINO
```

Este comando monta la imagen `imagen.bin` en el directorio `DESTINO`. Luego se pueden copiar archivo y directorios a la carpeta `DESTINO`, estos van a quedar guardados en la imagen. Al finalizar, se debe desmontar la imagen:

```
sudo umount DESTINO
```

Opción B: utilizando `mtools`.

```
mcopy -i imagen.bin FUENTE ::/DESTINO
```

Este comando copia `FUENTE` en el directorio `DESTINO` dentro de la imagen `imagen.bin`.

```
mmd -i imagen.bin DIRECTORIO
```

Este comando crea el directorio `DIRECTORIO` dentro de la imagen `imagen.bin`

Hay más comandos: `mcd mcopy mdeltree mdir minfo mlabel mrd mread mmove mren mshowfat mwrite`.
Más información: `http://www.gnu.org/software/mtools/intro.html`.

2. Enunciado

Ejercicios: El objetivo de este trabajo práctico es realizar una aplicación que realice las siguientes operaciones sobre una imagen de un medio de almacenamiento:

- Imprimir por pantalla la información sobre la imagen y el sistema de archivos.
- Listar los contenidos de un directorio dado.
- Calcular el tamaño total de los archivos en la imagen a partir de un directorio dado.
- Extraer un archivo dado de la imagen.

Suponiendo que la aplicación se llama `tpcopado`, tiene que soportar los siguientes parámetros:

- `tpcopado -v IMAGEN`: imprime la información de la imagen.
- `tpcopado -l DIRECTORIO IMAGEN`: lista los archivos en DIRECTORIO.
- `tpcopado -s DIRECTORIO IMAGEN`: suma el tamaño de todos los archivos en DIRECTORIO y sus subdirectorios.
- `tpcopado -e ARCHIVO IMAGEN`: Extrae ARCHIVO.

La aplicación debe estar programada en lenguaje ensamblador. Sólo es posible realizar en lenguaje C la función `main` para procesar la línea de comandos y ejecutar las funciones que correspondan.

Para abrir la imagen del medio de almacenamiento, recomendamos que utilicen la función `mmap` para generar un mapeo a memoria del archivo. En el ejercicio en el que deben extraer un archivo de la imagen, pueden utilizar write, pero es necesario abrir el archivo con el flag `O_CREAT` para que no falle en el caso de que el archivo a extraer no exista.

Informe

Debe reflejar el trabajo hecho para obtener el resultado, las decisiones tomadas (con el estudio de sus alternativas), las estructuras de datos usadas (con gráficos y/o dibujos si ayudan a clarificar), las pruebas que hayan hecho para tomar decisiones o al final para buscar errores en el producto final, etcétera. Debe contar como mínimo con los siguientes capítulos: introducción, desarrollo, discusión y conclusiones. Estar estructurado top-down o sea leyendo la introducción se debe saber qué se hizo y cuáles son las partes más importantes. Después de leer los primeros capítulos se debe saber cada cosa que se hizo y como se hicieron las más importantes. En el resto de los capítulos se debe poder leer el detalle de todo lo hecho.

Además, el informe debe incluir:

- Carátula con número del grupo y los nombres de los integrantes con número de libreta y email
- Manual del usuario
- Algunas imágenes con las que hayan probado y su correspondiente procesamiento.
- Instrucciones para el corrector, por ejemplo como ensamblar los archivos fuente para obtener el ejecutable (Makefile).
- Lista de todos los archivos entregados.

El informe se evalúa de manera independiente del código. Puede reprobarse el informe y en tal caso deberá ser reentregado para aprobar la materia.

Entrega

La fecha de entrega de este trabajo es martes 4 de mayo, en el horario de clase (de 17 a 22 hs). La entrega se realizará en un CD que debe incluir, los ejecutables, todos los archivos fuentes necesarios para crearlos, imágenes de prueba con su correspondiente procesamiento y el informe en `pdf`. Si desarrollaron prototipos en lenguaje C para resolver el trabajo primero en alto nivel, deben entregarlos e incluir los resultados obtenidos en el informe. Para ordenar la entrega se deben crear las siguientes carpetas en el CD: `src`, `exe`, `enunciado`, `informe`, `resultados`.

3. Anexo: cambios en el enunciado

3.1. Versión 2

3.1.1. Introducción teórica - Estructura - Área de directorio y datos - Se agrega:

Notas sobre el campo *nombre*:

El primer *byte* es tratado de forma especial. Estas son las reglas que se aplican:

`0x00` se interpreta como “detener la búsqueda, no hay mas entradas en este directorio”.

`0x20` no debe existir.

`0xe5` se interpreta como una entrada libre.

3.2. Versión 3

3.2.1. Introducción teórica - Estructura - FAT - Se agrega:

Notas sobre la numeración de los clústeres:

- Los clústeres se numeran del 0 en adelante.
- El primer clúster de datos tiene el índice 2.
- Las primer entrada de la FAT indica el tipo de medio. Debe ser el mismo que se indica en el sector de booteo.
- La segunda entrada de la FAT indica el estado. `0xFFFF` indica que está limpia. `0xF7FF` indica que esta montada o en uso.

3.2.2. Introducción teórica - Estructura - Área de directorio y datos - Notas sobre las entradas de directorio - Se agrega:

Con el paso del tiempo, el sistema FAT16 original se modificó para soportar nombres largos. Para ello, las entradas de directorio se empezaron a utilizar para algo mas que representar archivos y directorios. Hay entradas que contienen el nombre en formato *Unicode* y un campo que indica a que entrada del directorio corresponde ese nombre largo. Para no confundirlas con entradas de archivo o directorios, los valores del campo *atributos* y *primer clúster del archivo* tienen los valores `0x0F` y `0x0000` respectivamente. Valores que no tienen sentido dentro de una entrada de directorio. Para el objetivo de este trabajo práctico no es necesario tener en cuenta los nombres largos. Estas entradas deben ser obviadas.

3.2.3. Introducción teórica - Estructura - Área de directorio y datos - Byte de atributos - Se agrega:

El bit *archivo* no indica que es un archivo. Este se utiliza en sistemas de backup. Para determinar si es archivo o directorio, hay que ver el bit *directorio*.

3.2.4. Introducción teórica - Funciones - `open` - Se agrega:

El parámetro `mode` debe ser especificado si se usa el flag `O_CREAT`. Se utiliza para indicar los permisos con los que se va a crear el archivo. Puede ser una combinación de los siguientes:

- `S_IRWXU`: Lectura/Escritura/Ejecución del *owner*.
- `S_IRUSR`: Lectura del usuario.
- `S_IWUSR`: Escritura del usuario.
- `S_IXUSR`: Ejecución del usuario.
- `S_IRWXG`: Lectura/Escritura/Ejecución del grupo.
- `S_IRGRP`: Lectura del grupo.
- `S_IWGRP`: Escritura del grupo.

- `S_IXGRP`: Ejecución del grupo.
- `S_IRWXO`: Lectura/Escritura/Ejecución de otros.
- `S_IROTH`: Lectura de otros.
- `S_IWOTH`: Escritura de otros.
- `S_IXOTH`: Ejecución de otros.

3.2.5. Introducción teórica - Funciones - Se agrega:

```
ssize_t write(int fd, const void *buf, size_t count);
```

Esta función escribe `count` bytes en el archivo descripto por `fd` obtenidos a partir de `buf`. Devuelve la cantidad de bytes escritos.

3.2.6. Enunciado - Ejercicios - Se cambia:

Texto anterior

Para abrir la imagen del medio de almacenamiento, recomendamos que utilicen la función `mmap` para generar un mapeo a memoria del archivo. En el ejercicio en el que deben extraer un archivo de la imagen, pueden utilizar mmap, pero es necesario abrir el archivo con el flag `O_CREAT`.

Texto modificado

Para abrir la imagen del medio de almacenamiento, recomendamos que utilicen la función `mmap` para generar un mapeo a memoria del archivo. En el ejercicio en el que deben extraer un archivo de la imagen, pueden utilizar write, pero es necesario abrir el archivo con el flag `O_CREAT` para que no falle en el caso de que el archivo a extraer no exista.